DTRACK: A System to Predict and Track Internet Path Changes

Ítalo Cunha, Renata Teixeira, Darryl Veitch, *Fellow, IEEE*, and Christophe Diot, *Fellow, ACM*

***Abstract*—In this paper, we implement and evaluate a system that predicts and tracks Internet path changes to maintain an up-to-date network topology. Based on empirical observations, we claim that monitors can enhance probing according to the likelihood of path changes. We design a simple predictor of path changes and show that it can be used to enhance probe targeting. Our path tracking system, called DTRACK, focuses probes on unstable paths and spreads probes over time to minimize the chances of missing path changes. Our evaluations of DTRACK with trace-driven simulations and with a prototype show that DTRACK can detect up to three times more path changes than traditional traceroute-based topology mapping techniques.**

***Index Terms*—Path changes, topology mapping, tracking.**

I. Introduction

SYSTEMS that detect Internet faults [1], [2] or prefix hijacks [3] require frequent measurements of Internet paths, often taken with traceroute. Topology mapping techniques periodically issue traceroutes and then combine observed links into a topology [4]–[6]. Content distribution networks continuously monitor paths and their properties to select the "best" content server for user requests [7]. Similarly, overlay networks monitor IP paths to select the best overlay routing [8]. In all these examples, a source host issues traceroutes to a large number of destinations with the hope of tracking paths as they change.

The classical approach of probing all paths equally, however, has practical limits. First, sources have a limited probing capacity (constrained by source link capacity and CPU utilization), which prevents them from issuing traceroutes frequently enough to observe changes on all paths. Second, Internet paths are often stable [9]–[11], so probing all paths at the same frequency wastes probes on paths that are not changing while missing changes in other paths. Finally, many paths today traverse routers that perform load balancing [12]. Load balancing creates multiple simultaneous paths from a source to a given destination. Ignoring load balancing leads to traceroute errors and misinterpretation of path changes [11]. Accurately discovering all paths under load balancing, however, requires even more probes [13].

This paper shows that a monitor can attempt to optimize probing to track path changes more efficiently than classical probing given the same probing capacity. We develop DTRACK, a system that separates the tracking of path changes into two tasks: *path change detection* and *path remapping*. DTRACK only remaps, i.e., remeasures, a path once a change is detected. Path remapping uses Paris traceroute's multipath detection algorithm (MDA) [13] because it accurately discovers all paths under load balancing. The key novelty of this paper is to design a probing strategy that predicts the paths that are more likely to change and adapts the probing frequency accordingly. We make two main contributions.

Investigate the Predictability of Path Changes: We use traceroute measurements collected from 70 PlanetLab nodes over five weeks to train models of path changes. We use RuleFit [14], a supervised machine learning technique, to identify the features that help predict path changes and to act as a benchmark. We show that route prevalence, number of past route changes, number of times a route appears in the past, and route age are the best predictors of path changes (Section III). RuleFit is too complex to be used online. Hence, we develop a model to predict path changes, called NN4, based on the K nearest-neighbor scheme, which can be implemented efficiently and is as accurate as RuleFit (Section IV). We find that prediction is inherently difficult. Even though NN4 is not highly accurate, it is effective for tracking path changes, as it can predict paths that are likely to change in the short term.

Probing strategy to Track Path Changes: We use NN4 to optimize the allocation of probes to paths in DTRACK (Section V). DTRACK adapts path sampling rates to minimize the number of missed changes based on NN4's predictions. For each path, it sends a single probe per sample in a temporally striped form of traceroute. Our trace-driven simulations show that, for the same probing budget used by DIMES [15], DTRACK misses 73% less path changes than traditional traceroute-based mapping techniques and detects 93% of the path changes in the traces. Our measurements in the real-world deployment show that DTRACK detects three times more path changes than the state-of-the-art for the probing budget used by DIMES [15].

Manuscript received March 22, 2013; accepted May 24, 2013; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor J. Wang. Date of publication July 12, 2013; date of current version August 14, 2014. This work was supported by the European Community's Seventh Framework Programme (FP7/2007-2013) No. 223850 (Nano Data Centers) and the ANR project C'MON, and was also supported in part by the Australian Research Council's Discovery Projects funding scheme under Project No. DP120102834. A previous version of this work was published in the Proceedings of ACM SIGCOMM, Toronto, ON, Canada, August 15–19, 2011. This version simplifies the discussion on path change prediction, evaluates an implementation of DTRACK using PlanetLab, and covers recent related work.

I. Cunha is with the Universidade Federal de Minas Gerais, Belo Horizonte 31270-010, Brazil (e-mail: cunha@dcc.ufmg.br).

R. Teixeira is with UPMC Sorbonne Universités and CNRS, Paris 75015, France (e-mail: renata.teixeira@lip6.fr).

D. Veitch is with the Department of Electrical and Electronic Engineering, University of Melbourne, Melbourne, Vic. 3010, Australia (e-mail: dveitch@unimelb.edu.au).

C. Diot is with Technicolor, Issy-les-Moulineaux 92443, France (e-mail: christophe.diot@technicolor.com).

Digital Object Identifier 10.1109/TNET.2013.2269837

II. Definitions, Data, and Metrics

In this section, we define key concepts and present the dataset we use. We establish the low-level path prediction goals that underlie our approach to path tracking, and then present a spectrum of candidate data features we exploit to that end.

A. Virtual Paths and Routes

Following Paxson [9], we use *virtual path* to refer to the connectivity between a fixed source (here a monitor) and a destination d. At any given time, a virtual path is realized by a route that we call the *current route*. Since routing changes occur, a virtual path can be thought of as a continuous time process $P(t)$ that jumps between different routes over time.

A *route* can be *simple*, consisting of a sequence of IP interfaces from the monitor toward d, or *branched*, when one or more *load-balancing* routers are present, giving rise to multiple overlapping sequences (branched routes are called "multipaths" in [13]). A route can be a sequence that terminates before reaching d. This can occur due to routing changes (e.g., transient loops), or the absence of a complete route to the destination. By *route length*, we mean the length of its longest sequence, and we define the *edit distance* between two routes as the minimum number of interface insertions, deletions, and substitutions needed to make the IP interface sequences of each route identical. In the same way, we can define *autonomous system (AS) length* and *AS edit distance* for a general route.

Let a virtual path P be realized by route r at time t, i.e., $P(t) = r$. Suppose that the path will next jump to a new route at time t_d, and last jumped to the current route r at time t_b. Then, the *age* of this instance of route r is $A(r) = t - t_b$, its *residual life* is $L(r) = t_d - t$, and its *duration* is $D(r) = A(r) + L(r) = t_d - t_b$. Typically, as we have just done, we will write $A(r)$ instead of $A(P(t))$, and so on, when the context makes the virtual path, time instant, and hence route instance, clear.

In practice, we measure virtual paths only at discrete times, resulting effectively in a sampling of the process $P(t)$. A change can be detected whenever two consecutive path measurements differ, however the full details of the evolution of the virtual path between these samples is unknown, and many changes may be missed. For simplicity, unless stated otherwise, by *(virtual) path change*, we mean a change observed in this way. The change is deemed to have occurred at the time of the second measurement. Hence, the measured age of a route instance is always zero when it is first observed. This conservative approach underestimates route age with an error smaller than the intermeasurement period.

B. Dataset

For our purposes, an ideal dataset would be a complete record of the evolution of virtual paths, together with all sequences of IP interfaces for each constituent route. Real-world traces are limited both in the frequency at which each virtual path can be sampled and the accuracy and completeness of the routing information obtained at each sample. In particular, the identification of the multiple IP interface sequences for branched routes requires a lot of probes [13] and takes time, reducing the frequency at which we can measure virtual paths. For this identification, we use Paris traceroute's Multipath Detection Algorithm (MDA) [13]. MDA provides strong statistical guarantees for complete route discovery in the presence of an unknown number of load balancers. It is therefore ideal for reliable change detection, but is conservative and can be expensive in probe use (see Section V-D).

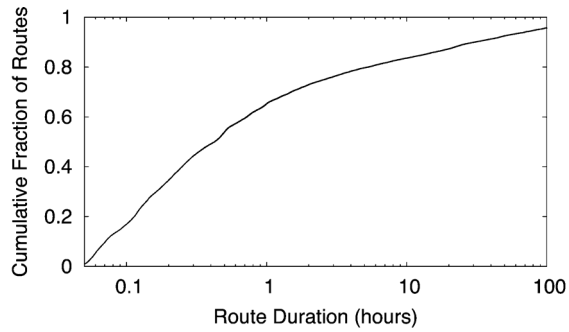


Fig. 1. Distribution of all route durations in the dataset.

We address the above limitations using traces collected with *FastMapping* [11]. FastMapping measures virtual paths with a modified version of Paris traceroute [12] that sends a single probe per hop. Whenever a new IP interface is seen, FastMapping remeasures the route using MDA. This way, the frequency at which it searches for path changes is high, but when a change is detected, the new route is mapped out thoroughly.

We use a publicly available dataset collected from 70 PlanetLab hosts during five weeks starting September 1, 2010 [11]. Each monitor selects 1000 destinations at random from a list of 34 820 randomly chosen reachable destinations. Each virtual path is measured every 4.4 min on average. We complement the dataset with IP-to-AS maps built from Team Cymru [16] and UCLA's IRL [17]. Although almost all monitors are connected to academic networks, the destinations are not. As such, this dataset traverses 7842 ASs and covers 97% of large ASs [17].

We lack ground truth about path changes, and the FastMapping dataset may miss changes. However, all changes that the dataset captures are real. Fig. 1 shows the distribution of all route durations in the dataset. It is similar to Paxson's findings that most routes are short-lived.

C. Prediction Goals and Error Metrics

We study three kinds of prediction:

1) prediction $\hat{L}(r)$ of the residual lifetime $L(r)$ of a route $r = P(t)$ of some path observed at time t;
2) prediction $\hat{N}_\delta(P)$ of the number of changes in the path occurring in the time interval $[t, t+\delta]$;
3) prediction, via an indicator function $\hat{I}_\delta(r)$, of whether the current route will change in the interval $[t, t+\delta]$ ($I_\delta(r) = 1$), or not ($I_\delta(r) = 0$).

In the case of residual lifetime, we measure the relative prediction error $E_L(r) = (\hat{L}(r) - L(r))/L(r)$. This takes values in $[-1, \infty)$, with $E_L(r) = 0$ corresponding to a perfect prediction. For $\hat{N}_\delta$, we measure the absolute error $E_{N_\delta}(P) = \hat{N}_\delta(P) - N_\delta(P)$ because the relative prediction error is undefined whenever $N_\delta(P) = 0$. For $\hat{I}_\delta$, we measure the error E_{I_δ}, the fraction of time $\hat{I}_\delta(r) \neq I_\delta(r)$. This takes values in [0,1], with $E_{I_\delta} = 0.5$ corresponding to a random predictor.

D. Virtual Path Features

A virtual path predictor needs to determine and exploit those features of the path and its history that carry the most information about change patterns.

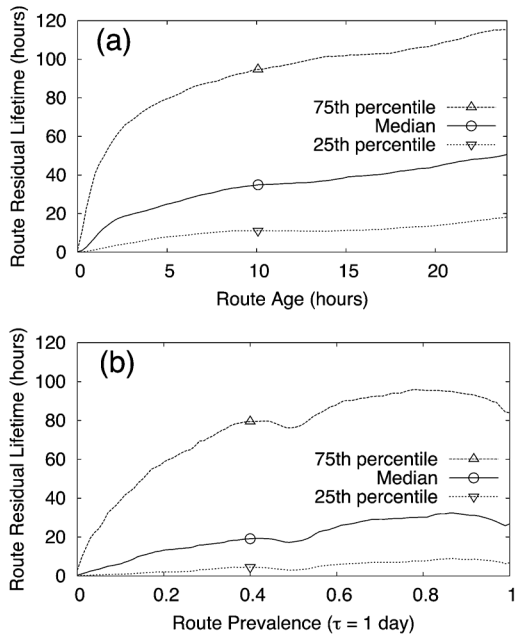


Fig. 2. Relationship between virtual path features and residual lifetime: residual lifetime as a function of (a) route age and (b) route prevalence.

Paxson characterized virtual path stability using the notions of route *persistence*, which is essentially route duration $D(r)$, and route *prevalence* [9], the proportion of time a given route is active. In the context of prediction, where only metrics derivable from past data are available, these two measures translate to the following two features of the route r that is current at time t: 1) the route age $A(r)$; 2) the (past) prevalence, the fraction of time r was active over the window $[t-\tau, t]$. We set the *timescale* τ to $\tau = \infty$ to indicate a window starting at the beginning of the dataset.

Route age and prevalence are important prediction features. Their utility is illustrated in Fig. 2(a) and (b), where the median, 25th, and 75th percentiles of route residual lifetimes are given as a function of the respective features (computed based on periodic sampling of all virtual paths in the dataset with period 5 min). In Fig. 2(a), for example, we observe that younger routes have shorter residual lifetimes than older routes, a possible basis for prediction. Similarly, Fig. 2(b) shows that when prevalence is measured over a timescale of $\tau = 1$ day, routes with lower prevalence are more likely to die young.

Although route age and prevalence are each useful for prediction, they are not sufficient as shown by the high variability in the data [wide spread of the percentiles in Fig. 2(a) and (b)]. Additional features are needed to do better. Our aim here is to define a spectrum of features broad enough to capture essentially all information computable from the dataset that may have predictive value. We do not know at this point which features are the important ones, nor how to combine them to make accurate predictions. This is a task we address in Section III.

We do not attempt to exploit spatial dependencies in this paper for prediction, although they clearly exist. For example, changes in routing tables impact multiple paths at roughly the same time. The reason is that including spatial network information in RuleFit requires one predictive feature per link in the network, which is computationally prohibitive. However, we *can* exploit spatial dependencies to improve path tracking efficiency in the probing scheme, as we detail in Section V-C.

TABLE I
SET OF CANDIDATE FEATURES UNDERLYING PREDICTION

CURRENT ROUTE
Route Age
Length, AS length
Number of load balancers (i.e., hops with multiple next-hops)
Indicator of whether the route reaches the destination
LAST CHANGE
Duration of the previous route
Length difference, AS length difference
Edit distance, AS edit distance
TIMESCALE-BASED (COMPUTED OVER $[t-\tau, t]$)
Prevalence of the current route
Average route duration
Standard deviation of route durations
Number of previous occurrences of the current route
Number of virtual path changes
EVENT-BASED
Times since the most recent occurrences of the current route
Num. of changes since the most recent occ. of the current route

Table I partitions all possible features into four categories:

1) Current route: characterize the current route and its state;
2) Last change: capture any nearest neighbor interactions;
3) Timescale-based: metrics measured over a given timescale;
4) Event-based: metrics defined in "event-time."

We use this scheme only as a framework to guide the selection of individual features. We aim to capture inherently different kinds of information and measures both of average behavior and variability. Only features that are computable based on the information in the dataset, together with available side-information (we use IP-to-AS maps), are allowed.

The last four features in the Timescale-based category allow us to identify virtual paths that are highly unstable and change repeatedly, as observed by previous work [9], [18], [19]. The features in the Event-based category may involve time, but are not defined based on a preselected timescale. Instead, they try to capture patterns of changes in the past, like oscillation between two routes. For computational reasons, we limit ourselves to looking up to the five most recent virtual path changes. In most cases, this is already sufficient to reach the beginning of the dataset. An overview of feature properties for the dataset is given in our previous work [20].

III. PREDICTION FOUNDATIONS

Our path tracking approach is built on the ability to predict (albeit imperfectly) virtual path changes. We seek a predictor based on an intuitive and parsimonious model rather than a black box. However, virtual path changes are characterized by extreme variability and are influenced by many different factors, making model building, and even feature selection, problematic. We employ RuleFit [14], a supervised machine learning technique, to bootstrap our modeling efforts. We use Rulefit for two main purposes. First, to comprehensively examine the spectrum of features of Table I in order to determine those that are most predictive. Second, to act as a benchmark representing in an approximate sense the best possible prediction when large (offline) resources are available for training.

A. RuleFit Overview

RuleFit [14] trains predictors based on rule ensembles. We choose it over other alternatives (against which it compares favorably) for two reasons: 1) it ranks features by their importance for prediction; 2) it outputs easy-to-interpret rules that allow an understanding of how features are combined. In other words, for a black box, it is a lighter shade of gray. We give a brief overview of RuleFit, referring the reader to the original paper for full details [14].

Rules combine one or more features into simple "and" tests. Let $\boldsymbol{x}$ be the feature vector in Table I and s_f a specified subset of the possible values of feature f. Then, a rule takes the form

$$r(\boldsymbol{x}) = \prod_f I(x_f \in s_f) \tag{1}$$

where $I(\cdot)$ is an indicator function. Rules take value one when features values fall inside their corresponding ranges, else zero.

RuleFit first generates a large number of rules using decision trees. It then trains a predictor of the form

$$\hat{\phi}(\boldsymbol{x}) = a_0 + \sum_k a_k r_k(\boldsymbol{x}) \tag{2}$$

where the vector $\boldsymbol{a}$ is computed by solving an optimization problem that minimizes the *Huber loss* (a modified squared prediction error robust to outliers) with an L1 penalty term. RuleFit also employs other robustness mechanisms—for example, it trains and tests on subsets of the training data internally to avoid overfitting.

Rule ensembles can exploit feature interdependence and capture complex relationships between features and prediction goals. Crucially, RuleFit allows rules and features to be ordered by their importance. *Rule importance* is the product of the rule's coefficient and a measure of how well it splits the training set

$$I_k = |a_k|\sqrt{s_k(1-s_k)}$$

where s_k is the fraction of points in the training set where $r_k(\boldsymbol{x}) = 1$. *Feature importance* is computed as the sum of the normalized importance of the rules where the feature appears

$$I_f = \sum_{k:f\in r_k} I_k/m_k \tag{3}$$

where m_k is the number of active features in r_k.

B. RuleFit Training Sets

RuleFit, like any supervised learning algorithm, requires a training set consisting of training points that associate features with the true values of metrics to be predicted. Here, a training point, say for residual lifetime, associates a virtual path at some time t, represented by the features in Table I, with the true residual lifetime $L(r)$ of the current route $r = P(t)$. Separate but similar training is performed for $N_\delta(P)$ and $I_\delta(r)$.

To limit the computational load of training, which is high for Rulefit, we control the total number of training points. For training point selection, first note that a given virtual path has a change history that is crucial to capture for good prediction of its future. We therefore build the required number of training points by extracting rich path change information from a subset of paths, rather than extracting (potentially very) partial information from each path. We retain path diversity through random path selection and the use of multiple training sets obtained through using different random seeds (at least five for each parameter configuration we evaluate).

For a given virtual path, we first include all explicit path change information by creating a training point for each entry in the dataset where a change was detected. However, such points all have (measured) current route age equal to zero (Section II-A), whereas when running live predictions in general are needed at any arbitrary time point, with arbitrary route age. To capture the interdependence of features and prediction targets on route age, we include additional synthetic points that do not appear in the dataset but that are functions of it. To achieve this, we discretize route age into bins and create a training point whenever the age of a route reaches a bin boundary. We choose bin boundaries as equally spaced percentiles of the distribution of route durations in the training set, as this adapts naturally to distribution shape. Using six bins as an example, we create training points whenever a route's age reaches 0 s, 3 min, 7 min, 24 min, 75 min, and 6.5 h. We note that route age inference error is less than the measurement round duration (4.4 min in the FastMapping dataset), which is smaller than bin sizes and has no noticeable impact on prediction accuracy.

C. Test Sets

Like training sets, test sets consist of test points that associate virtual path features with correct predictions. Unlike training sets, where the primary goal is to collect information important for prediction and where details may depend on the method to be trained, for test sets the imperative is to emulate the information available in the operational environment so that the predictor can be fairly tested and should be independent of the prediction method.

The raw dataset has too many points for use as a test set. To reduce computational complexity, we build test sets by sampling each virtual path at time points chosen according to a Poisson process, using the same sampling rate for each path. This corresponds to examining the set of paths in a neutral way over time, which will naturally include a diversity of behavior. For example, our test sets include samples inside bursts of path changes, as well as many samples from a very long-lived route, and rare events such as of an old route just before it changes.

We use an average per-path sampling period of 4 h, resulting in at least two orders of magnitude more test points than training points. We test each predictor against eight different test sets (from different Poisson seeds), for a total of 40 different training-test set combinations.

We ignore routes active at the beginning or the end of the dataset when creating training and test sets, as their duration, age, and residual lifetime are unknown. Similarly, we ignore all virtual path changes in the first τ hours of the dataset (if $\tau \neq \infty$) to avoid biasing timescale-dependent features.

D. RuleFit Configuration

We evaluate different RuleFit configurations to assess the limits of prediction accuracy and to choose default parameter

TABLE II
FEATURE IMPORTANCE ACCORDING TO RULEFIT

PATH FEATURE	IMPORTANCE
Prevalence of the current route ($\tau = 1$ day)	1.0
Num. of virtual path changes ($\tau = 1$ day)	.624
Num. of previous occ. of the current route ($\tau = 1$ day)	.216
Route age	.116
Times since most recent occs. of the current route	$\leq$ .072
Edit distance (last change)	.015
Duration of the previous route	.014
Standard deviation of route durations ($\tau = 1$ day)	.014
Length difference (last change)	.012
All other features	$\leq$ .010

values. We present a summary of our findings and refer the reader to our previous work for more details [20].

We study the impact of four parameters on prediction error: the number of rules generated during training, the number of age thresholds, the timescale τ, and the training set size. We find that increasing parameter values improves prediction accuracy, but improvement quickly flattens out. Our interpretation is that, after some point, additional data is redundant, and additional flexibility is unused as there is no extra information relevant for prediction. We set the default parameter values in the range where increasing their values does not improve prediction accuracy. We set the default number of rules to 200, the default number of age bins to six, the number of path changes in training sets to 200 000 (around 2.4% of the changes in the dataset), and the timescale τ to one day.

We justify our use of RuleFit as a benchmark for predicting changes, based on a given (incomplete) dataset, on three facts: 1) we provide RuleFit with a rich feature set; 2) RuleFit performs an extensive search of feature combinations to predict residual lifetimes; and 3) our evaluation shows that changing RuleFit's parameters from our default configuration is unlikely to improve prediction accuracy significantly. This is an empirical approach to approximately measure the limits to prediction using a given dataset. Determining actual limits would only be possible given information-theoretic or statistical assumptions on the data, which is beyond the scope of this paper.

E. Feature Selection

We compute feature importance with (3) and normalize using the most important feature. Table II shows features ordered by normalized importance averaged over 50 predictors for each of residual lifetime, number of changes, and I_δ.

Route prevalence is the most important feature, helped by its correlation with route age. It is clear why route prevalence alone is insufficient. Route prevalence cannot differentiate a young current route that occurred repeatedly in the time window of width τ, from a middle-aged current route, as both have intermediate prevalence values.

The second, third, and fourth most important features are the number of virtual path changes, the number of occurrences of the current route, and route age. Predicted residual lifetimes increase as route age and prevalence increase, but decrease as the number of virtual path changes and occurrences of the current route increase. Results for the number of changes and I_δ are similar.

The fifth most important feature is the times (first up to fifth) of the most recent occurrences of the current route. The low importance of this and the other event-based feature suggest that, contrary to our initial hopes, patterns of changes are too variable, or too rare, to be useful for prediction.

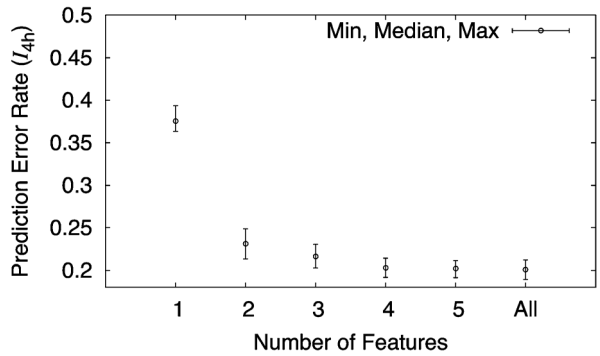


Fig. 3. $E_{4\mathrm{h}}$ for predictors trained with the most important features (test points with route age less than 12 h).

To evaluate more objectively the utility of RuleFit's feature importance measure, Fig. 3 shows $E_{I_{\delta=4\mathrm{h}}}$ for predictors trained with training sets containing only the top p features, for $p = 1$ to 5. The improvements in performance with the addition of each new feature are consistent with the importance rankings from Table II. Importantly, we see that the top four features generate predictors that are almost as accurate as those trained on all features. This will be crucial in what follows.

IV. NEAREST-NEIGHBOR PREDICTOR

We design and evaluate a simple predictor that is almost as accurate as RuleFit while overcoming its drawbacks, namely its slow and computationally expensive training, its difficult integration into other systems, and the lack of insight and control arising from its black box nature.

A. NN4: Definition

Our starting point is the observation that the top four features from Table II carry almost all of the usable information. Since virtual paths are so variable and the RuleFit models we obtained are so complex, simple analytic models are not serious candidates as a basis for prediction. We select a nearest-neighbor approach as it captures empirical dependences effectively and flexibly. Using only four features avoids the dimensionality problems inherent to such predictors [21] and allows for a very simple method, which we name NN4.

1) Method Overview: Like all nearest-neighbor predictors, we compute predictions for a virtual path with feature vector $\boldsymbol{x}$ based on training points with feature vectors that are "close" to $\boldsymbol{x}$. The first challenge is to define a meaningful distance metric. This is difficult as feature domains differ (prevalence is a fraction, the number of changes and previous occurrences are integers, and route age is a real), have different semantics, and impact virtual path changes differently.

To avoid the pitfalls of some more or less arbitrary choice of distance metric, we instead partition the feature space into four dimensional "cubes," or partitions, based on discretizing each feature. Discretization creates artifacts related to bin boundaries and resolution loss, however the great advantage is simplicity, and the retention of a meaningful notion of distance for each feature individually. To avoid rigid fixed bin boundaries, for each feature, we choose them as equally spaced percentiles of their corresponding distribution, computed over all virtual

path changes in the training set (as we did for route age in Section III-B).

We denote the partition containing the feature vector of path P at time t as $\mathcal{P}(P,t)$ or simply $\mathcal{P}(P)$. We predict the residual lifetime of $r = P(t)$ and the number of changes in the next δ interval as the averages of the true values of these quantities over all training points in the partition $\mathcal{P}(P)$

$$\hat{L}(r) = E[\{L(P_s(t_s))|s \in \mathcal{P}(P)\}]$$
$$\hat{N}_\delta(P) = E[\{N_\delta(P_s)|s \in \mathcal{P}(P)\}]$$

where training point s corresponds to the path P_s at time t_s. Similarly, we predict $\hat{I}_\delta(r) = 1$ if more than half the training points in $\mathcal{P}(P)$ change within a time interval δ

$$\hat{I}_\delta(r) = \lfloor E[\{I_\delta(P_s(t))|s \in \mathcal{P}(P)\}] + 0.5 \rfloor.$$

The cost of a prediction in NN4 is $O(1)$, while in RuleFit it is $O(r)$, where r is the number of rules in the model. NN4 can be easily implemented, while RuleFit is available as a binary module that cannot be accessed directly and requires external libraries. Finally, training time for RuleFit is orders of magnitude higher than for the NN4 predictor.

2) Training: To allow a meaningful comparison in our evaluation, each training for NN4 reuses the virtual paths of some RuleFit training set.

Consider a virtual path $P(t_s)$ chosen for training. As t_s progresses, the associated feature vector $\boldsymbol{x}(t_s)$ moves between the different partitions. For example, for long-lived routes, $\boldsymbol{x}(t_s)$ evolves toward the partition with 100% prevalence, zero changes, no previous occurrences, and the oldest age bin (before resetting to zero age, etc., when/if the path changes). We need to sample this trajectory in a way that preserves all important information about the changes in the three prediction goals (L, N_δ, I_δ). Just as in RuleFit, we need to supplement the changes that occur explicitly in the dataset with additional training points occurring in between change points. Here, we need to add additional samples to capture the diversity not only of age, but also the other three dimensions. In fact, we can do much better than a discrete sampling leading to a set of training time points. From the dataset, we can actually calculate when the path enters and exits the partitions it visits, its sojourn time in each, and the proportions of the sojourn time when a prediction goal takes a given value. For each partition (and prediction goal), we are then able to calculate the exact time-weighted average of the value over the partition. The result is a precomputed prediction for each partition traversed by the path that emulates a continuous-time sampling. Final per-partition predictions are formed by averaging over all paths traversing a partition.

3) Configuration: Apart from δ, the only parameter of our predictor is the number of bins b we use to partition each feature. We choose a shared number of bins for parsimony since, when studying each feature separately (not shown), the optimal point was similar for each. The tradeoff here is clear. Too few bins and distinct change behaviors important for prediction are averaged away. Too many bins and partitions contain insufficient training information resulting in erratic predictions. We found in Section III-D that six bins were sufficient for route age. We now examine all features together.

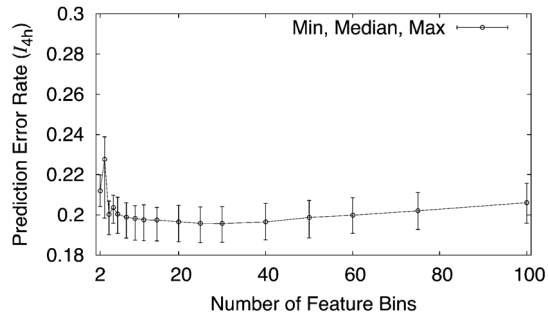


Fig. 4. Impact of the number of feature bins on prediction accuracy (test points with age $A < 12$ h).

Fig. 4 shows E_δ with $\delta = 4$ h as a function of b, restricting to test points with route age below 12 h where the b dependence is strongest. We see that values in [5, 20] achieve a good compromise. We use $b = 10$ in what follows.

B. NN4: Evaluation

We evaluate the prediction accuracy of NN4 and compare it to our operational benchmark, RuleFit, discovering in the process the limitations of this kind of prediction in general. For each method, we generate new training and test sets in order to test the robustness of the configurations determined above.

1) Predicting Residual Lifetime: Fig. 5(a) shows the distribution of $E_L(r)$, the relative error of $\hat{L}(r)$. An accurate predictor would have a sharp increase close to $E_L = 0$ (dotted line), but this is not what we see. Specifically, only 33.5% of the RuleFit and 31.1% of the nearest-neighbor predictions have $-0.5 \leq E_L \leq 1$ (see symbols on the curves). Predictions miss the true residual lifetimes by a significant amount around 70% of the time. As this is true not only of NN4 but also for Rulefit, we conjecture that accurate prediction of route residual lifetimes is too precise an objective with traceroute-based datasets. The error of NN4 is considerably larger than that of the benchmark, but is of the same order of magnitude.

It does not follow, however, that $\hat{L}(r)$ is not a useful quantity to estimate. Fig. 5(b) shows the marginal distribution of NN4 residual lifetime predictions as a function of true residual lifetime. Even though predictions are inaccurate, paths with smaller true residual lifetime [like the solid line in Fig. 5(b)] have predicted residual lifetimes lower than paths with larger true residual lifetime (dotted lines). This is enough to distinguish stable from unstable paths sufficiently accurately to bring important benefits to path tracking, as we show later.

2) Predicting Number of Changes: Fig. 6 shows the distribution of E_{N_δ}, the error of $\hat{N}_\delta$, for NN4 for all test points with route age less than 12 h. The errors for RuleFit are similar. Errors for test points in routes older than 12 h are significantly smaller (not shown) because a predictor can perform well simply by outputting "no change" ($\hat{N}_\delta = 0$). We focus here on the difficult case of $A < 12$ h.

Unlike residual lifetimes, the sharp increase near zero means most predictions are accurate. For example, 90.2% of test points have $-2 < E_{N_{4h}} < 2$, and accuracy increases for smaller values of δ. However, predicting the number of changes over long intervals such as 24 h cannot be done accurately. Note that simply guessing that $N_\delta = 0$ also works well for very small δ.

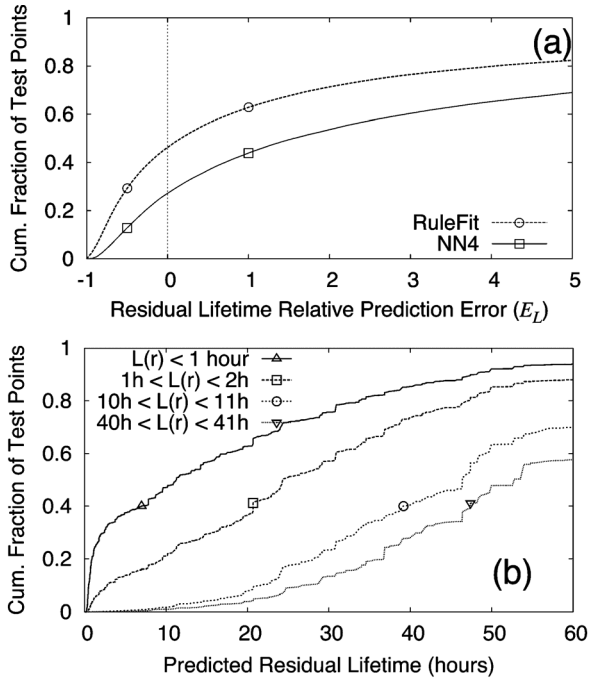


Fig. 5. Predicting residual lifetime (routes with age less than 12 h): (a) distribution of prediction error and (b) marginal distribution of NN4 predictions conditional on a residual lifetime range.

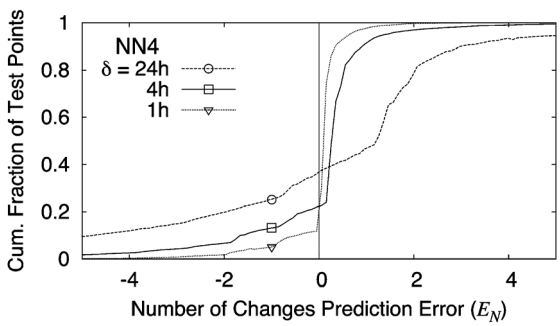


Fig. 6. Distribution of the number of changes prediction error E_N (routes with age $A < 12$ h).

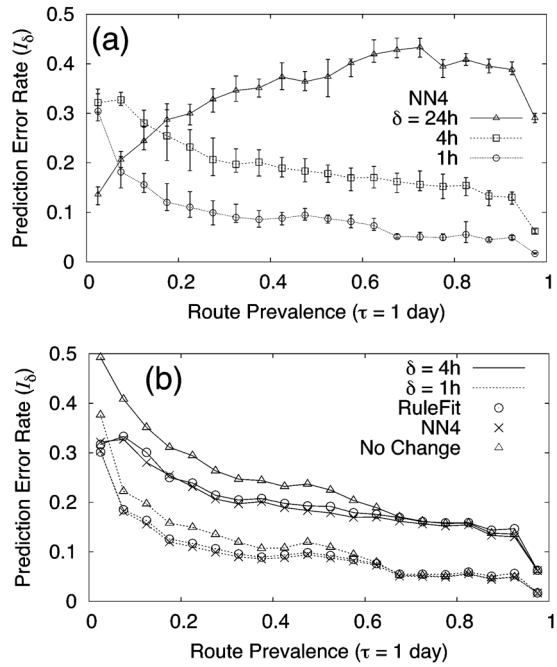


Fig. 7. E_{I_δ} as a function of route prevalence for various values of prediction horizon δ: (a) NN4; (b) comparison to RuleFit.

Although N_δ is a less ambitious target than L, it remains difficult to estimate from traceroute-type data. Again, however, prediction is sufficiently good to bring important tracking benefits.

3) Predicting a Change in Next δ Interval: We now study whether the current route of a given path will change within the next time interval of width δ. We expect I_δ to be easier to predict than L or N_δ.

Fig. 7(a) shows NN4's prediction error as a function of route prevalence for δ between 1 h and 1 day (results for RuleFit are very similar and are omitted for clarity). We group route prevalence into fixed-width bins and compute the error from all test points falling within each bin (these bins are distinct from the constant-probability bins underlying NN4's partitions). For each bin, we show the minimum, median, and maximum error among the 40 training and test set combinations. Such a breakdown is very useful as it allows us to resolve where prediction is more successful or more challenging. For example, since routes with prevalence 1 are very common, a simple global average over all prevalence values would drown out the results from routes with prevalence below 1.

First consider the results for $\delta = 1$ and 4 h. The main observation is that error drops with increasing prevalence. This is mainly because routes with high prevalence are unlikely to change, and a prediction of "no change" ($\hat{I}_\delta(r) = 0$), which the predictors in fact output increasingly often, becomes increasingly valid as prevalence increases. We also see that, for all prevalence values, error is lower for smaller δ. This makes intuitive sense since prediction further into the future is in general more difficult. More precisely, the probability that a route will change in a time interval δ decreases as δ decreases, and predictors exploit this by predicting "no change" more often.

The situation is more complex when $\delta = 24$ h, with errors beginning low and increasing substantially before finally peaking and then decreasing at very high prevalence. This happens because for larger values of δ, routes with low prevalence have a high probability of changing. Predictors exploit this and output $I_{24\text{h}}(r) = 1$ more often (in fact, more than 80% of the time for paths with prevalence under 0.2). Prediction error is highest at intermediate prevalence values, as these routes have a probability close to 50% of changing in the next 24 h. Finally, prediction error decreases for routes with high prevalence: Ss routes become stable, the same mechanism noted above for smaller δ kicks in.

In summary, prediction is easiest when δ is small and prevalence is high. This is a promising result as most Internet routes are long-lived and have high prevalence; moreover, applications like topology mapping need to predict changes within short time intervals. Fig. 7 shows that NN4 predicts I_δ for $\delta \leq 4$ h with at least 67% accuracy, with errors ultimately falling to just a few percent as route prevalence increases and as δ decreases.

We now provide a comparison against Rulefit, focusing on small to medium δ. Fig. 7(b) shows that NN4 and RuleFit have equivalent prediction accuracy across all values of prevalence. In fact, NN4 is marginally (up to 2%) better here, where we use the default Rulefit configuration. Their performance is close to identical when using the more generous RuleFit configuration (see Section III) with 500 rules and 12 age bins.

The plot also shows results for a simple baseline predictor that always predicts $\hat{I}_\delta(r) = 0$ (no change). Our predictor is better for routes with prevalence smaller than 0.7, which are more likely to change than not, but for high-prevalence routes, all predictors predict "no change" and are equivalent. For routes with prevalence below 0.7, NN4 reduces the baseline predictor's $E_{I_{4\mathrm{h}}}$ from 0.296 to 0.231 (22%), and $E_{I_{1\mathrm{h}}}$ from 0.163 to 0.131 (20%).

We have tested the sensitivity of our findings to training and test sets, monitor choice, and overall probing rate and found it to be low to very low.

V. Tracking Virtual Path Changes

We now apply our findings to the problem of the efficient and accurate tracking of a set of virtual paths over time. We describe and evaluate our tracking technique, DTRACK.

A. DTRACK *Overview*

Path tracking faces two core tasks: path change detection (how best to schedule probes to hunt for changes) and path remapping (how to update the topology after a change). For the latter, inspired by FastMapping [11], DTRACK uses Paris traceroute's MDA to accurately measure the current route of monitored paths both at start up and after any detection of change. This is vital since confusing path changes with load-balancing effects makes "tracking" meaningless.

For change detection, DTRACK is novel at two levels.

- *Across paths*: Paths are given dedicated sampling rates guided by NN4 to focus effort where changes are more likely to occur. Without this, probes are wasted on paths where nothing is happening.
- *Within paths*: A path "sample" is a single probe rather than a full traceroute, whose target interface is carefully chosen to combine the benefits of Paris Traceroute over time with efficiencies arising from exploiting links shared between paths. This allows changes to be spotted more quickly.

DTRACK monitors operate independently and use only locally available information. Each monitor takes three inputs—a trained predictor of virtual path changes, a set $\mathcal{D}$ of virtual paths to monitor, and a probing budget—and consists of three main routines: sampling rate allocation, change tracking, and change remapping. When a change is detected in a path through sampling, this path is remapped, and sampling rates are recomputed for all paths. A probing budget is commonly used to control network resource use [4], [15].

B. Path Sampling Rate Allocation

For each path p in $\mathcal{D}$, DTRACK uses NN4 to determine the rate λ_p at which to sample it. Sampling rates are updated whenever there is a change in the predictions, i.e., whenever any virtual path's feature vector changes its NN4 partition. This can happen as a result of a change detection or simply route aging.

We constrain sampling rates to the range $\lambda_{\min} \le \lambda_p \le \lambda_{\max}$. Setting $\lambda_{\min} > 0$ guarantees that all paths are sampled regularly, which safeguards against poor predictions on very long-lived paths. An upper rate limit is needed to avoid probes appearing as an attack ($\lambda_{\max}$ implements the "politeness" of the tracking method [22]).

Based on the monitor's probe budget of B probes per second, a sampling budget of B_s samples per second for the change detection alone can be derived (see Section V-D). To be feasible, the rate limits must obey $\lambda_{\min} \le B_\mathrm{s}/|\mathcal{D}| \le \lambda_{\max}$, where $|\mathcal{D}|$ is the number of paths.

We now describe three allocation methods for the sampling rates λ_p. The first two are based on residual life, and the third minimizes the number of missed changes.

Residual lifetime allocation (RL)*:* Since $1/L$ is precisely the rate that would place a sample right at the next change, allocating sampling rates proportional to $1/\hat{L}$ is a natural choice. We will see that despite the poor accuracy of $\hat{L}$ found before, this is far better than the traditional uniform allocation. We define rates to take values in $\lambda_p \in \{\lambda_{\max}, a/\hat{L}(p), \lambda_{\min}\}$ and require that $\lambda_p \ge \lambda_q$ if $L(p) < L(q)$ and $\lambda_{\min} \le \lambda_p \le \lambda_{\max}$ for all p, where a is a renormalization constant that respects $\sum_p \lambda_p = B_\mathrm{s}$ while minimizing the number of paths with rates clipped at $\lambda_{\min}$ or $\lambda_{\max}$.

We define two variants depending on the definition of $\hat{L}(p)$.

(RL): $\hat{L}(p)$ is estimated by NN4.

(RL-AGE): $\hat{L}(p)$ is predicted simply as the average residual lifetime of all route instances in the dataset with duration larger than $A(r)$, i.e.,

$$\hat{L}'(r) = \mathrm{E}[\{D(s) | s \in \mathcal{R} \text{ and } D(s) > A(r)\}] - A(r)$$

where $\mathcal{R}$ is the set of all route instances in the dataset.

Finally, for comparison, we add an oracular method that knows the true $L(p)$ and is not subject to rate limits:

(RL-ORACLE): $\lambda_p = a'/L(p)$ where $a' = B_\mathrm{s} \sum_q 1/L(q)$.

*Minimizing Missed Changes (*MINMISS*, used in* DTRACK*):* We use a Poisson process as a simple model for when changes occur. With this assumption, we are able to select rates that minimize the expected number of missed changes over the prediction horizon δ. This combines prediction of N_δ with a notion of sampling more where the pay off is higher. The rate $\mu_c(p)$ of the Poisson change process is estimated as $\mu_c(p) = \hat{N}_\delta(p)/\delta$.

We idealize samples as occurring periodically with separation $1/\lambda_p$. By the properties of a Poisson process, the changes falling within successive gaps between samples are i.i.d. Poisson random variables with parameter $\mu = \mu_c(p)/\lambda_p = \hat{N}_\delta/(\delta\lambda_p)$. Let C be the number of changes in a gap and M the number of these missed by the sample at the gap's end. It is easy to see that $M = \max(0, C-1)$ since a sample can see at most one change (here we assume that there is at most one instance of any route in the gap). The expected number of missed changes in a gap is then

$$\begin{aligned}\mathrm{E}[M(\mu)] &= \sum_{m=0}^{\infty} m \Pr(M=m) = \sum_{m=1}^{\infty} m \Pr(C=m+1) \\ &= e^{-\mu} \sum_{m=1}^{\infty} \frac{m\mu^{m+1}}{(m+1)!} = \mu - 1 + e^{-\mu}. \end{aligned} \tag{4}$$

Summing over the $\delta\lambda_p$ gaps, we compute the sampling rates as the solution of the following optimization problem:

$$\min_{\{\lambda_p\}} : \sum_p \delta\lambda_p(\mu - 1 + e^{-\mu}) = \sum_p \hat{N}_\delta + \delta\lambda_p(e^{-\hat{N}_\delta/(\delta\lambda_p)} - 1)$$
$$\text{such that } \sum_p \lambda_p = B_\mathrm{s}, \lambda_{\min} \le \lambda_p \le \lambda_{\max} \qquad \forall p.$$

We also evaluate I_δ as the basis of rate allocation, but as it is inferior to MINMISS, we omit it for space reasons.

Implementation: Path sampling in DTRACK is controlled to be "noisily periodic." As pointed out in [23], strictly periodic sampling carries the danger of phase locking with periodic network events. Aided by the natural randomness of round-trip times, our implementation ensures that sampling has the noise in intersample times needed to avoid such problems [24].

DTRACK maintains a first-in–first-out (FIFO) event queue that emits a sample (in fact a probe) every $1/B_{\rm s}$ s on average. Path p maintains a timer $T(p)$. When $T(p) = 0$, the next sample request is appended to the queue and the timer is reset to $T(p) = 1/\lambda_p$. Whenever DTRACK updates sampling rates, the timers are rescaled as $T_{\rm new}(p) = T_{\rm old}(p)\lambda_{{\rm old},p}/\lambda_{{\rm new},p}$. Path timers are staggered at initialization by setting $T(p_i) = i/B_{\rm s}$, where i indexes virtual paths.

C. In-Path Sampling Strategies

We define a *sample* of a path as a measurement of its current route obtained with one or more probes. At one extreme, a sample could correspond to a detailed route mapping using MDA. However, when checking for route changes rather than mapping from scratch, this approach is too expensive. We now investigate a number of alternatives that are less rigorous (a change may be missed) but cheaper (require less probes).

In all strategies below, the sample is load-balancing aware. We make use of the flow-id to interface mapping, established by the last full MDA, to target interfaces to test in an informed and strategic way. Thus, although a single sample takes only a partial look at a path and may miss a change, it will not flag a change where none exists and can still cover the entire route through multiple samples over time.

Per-Sequence: A single interface sequence from the route is selected. A sample probes the sequence's interfaces in order from the monitor to the destination using a single probe per interface. Subsequent samples select other sequences in some order until all are sampled and the route is covered, before repeating. This strategy gives detailed information but uses many probes in a short space of time. FastMapping has a similar strategy, only it probes a single sequence repeatedly rather than looping over all sequences.

Per-Probe: The interface testing schedule is exactly as for per-sequence. However, a sample probes a single interface. Thus, the probing of each sequence (and ultimately each interface in the route) is spread out over multiple samples.

The above methods treat each path in isolation, but paths originated at a single monitor often have shared links. Doubletree [25] and Tracetree [6] assume that the topology from a monitor to a set of destinations is a tree. They reduce redundant probes close to the monitor by doing backwards probing (from the destinations back to the monitor). Inspired by this approach, we describe methods that exploit spatial information, namely knowledge of shared links, to reduce wasteful probing while remaining load-balancing-aware. We define a *link* as a pair of consecutive interfaces found on some path, which can be thought of as a set of links. Many paths may share a given link.

Per-Link: A sample sends a single probe targeting the far interface of the least recently sampled link in the path. The per-link sample sharing scheme means that the timestamp recording the last sampling of a given link is updated by *any* path that contains it. The result is that a given path does not have to sample shared links as often, instead focusing more on links near the destination. Globally over all links, the allocation of probes to links becomes closer to uniform.

Per-Safelink: As for per-link, except that a shared link only triggers sample sharing when in addition an entire subsequence, from the monitor down to the interface just past the link, is shared. Any method that tries to increase probe efficiency through knowledge of how paths share interfaces can fail. This happens when a change occurs at a link (say ℓ) in some path p, but the monitor probes ℓ using a path other than p, for which ℓ has not changed. To help reduce the frequency of such events, per-link strengthens the definition of sharing from an interface to a link, and per-safelink expands it further to a subsequence.

Finally, for comparison, we add an oracular method.

Per-Oracle: A sample sends a single probe, whose perfect interface targeting will always find a change if one exists.

D. Evaluation Methodology

We describe how we evaluate DTRACK and compare it to other tracking techniques.

1) Setting Probe Budgets: The total probe budget B is the sum of a *detection budget* $B_{\rm d}$ used in sampling for change detection and a *remapping budget* or cost $B_{\rm r}$ for route remapping. Let the number of probes per sample be denoted by $n_{\rm sam}$, where *sam* is one of sampling strategies above. The total budget (in probes per second) can be written as

$$B = B_{\rm d} + B_{\rm r} = n_{\rm sam}B_{\rm s} + N_{\rm r} \cdot \overline{\rm MDA} \tag{5}$$

where $\overline{\rm MDA}$ is the average number of probes in a remapping, and $N_{\rm r}$ is the average number of remappings per second.

When running live in an operational environment, typical estimates of $N_{\rm r}$ and $\overline{\rm MDA}$ can be used to determine $B_{\rm s}$ based on the monitor parameter B. Our needs here are quite different. For the purposes of a fair comparison, we control $B_{\rm d}$ to be the same for all methods, and set DTRACK sampling rates $B_{\rm s} = B_{\rm d}/n_{\rm sam}$, where *sam* is the sampling strategy in use. Our principal measure of success is the detection of as many changes as possible. It does not make sense in this context to give each method the same total budget B, as it includes the remapping cost. The remapping cost is essentially just proportional to the number of changes detected; it would be contradictory to focus on $B_{\rm r}$ and to view its increase as a failing. The total budget B is important for the end system, but not of central interest for assessing detection performance. We provide some system examples based on equal B.

The default MDA parameters are very conservative, leading to high probe use. However, it is stated that much less conservative parameters can be used with little ill effect [13]. In this paper, we use default parameters for simplicity since the change detection performance is our main focus.

2) Performance Metrics: We evaluate two performance metrics for path tracking techniques: the fraction of missed virtual path changes, and the change detection delay.

A change can be missed through a sample failing to detect a change or because of undersampling. We give two examples of the latter. If a path changes from r_1 to r_2 and back to r_1 before a sample, then the tracking technique will miss two changes and think that the path is stable between the two samples. If instead the path changes from r_1 to r_2 to r_3, then tracking will detect a change from r_1 to r_3. For each detected change (and only for

detected changes), we compute the detection delay as the time of the detection minus the time of the last true change.

3) Alternative Tracking Techniques: We compare DTRACK against two other techniques: FastMapping [11] (Section II-B) and Tracetree [6] (Section V-C).

Comparing Tracetree against FastMapping and DTRACK is difficult because Tracetree assumes a tree topology and is also oblivious to load balancing. As such, Tracetree detects many changes that do not correspond to any real change in any path. To help quantify these false positives and to make comparison more meaningful, in addition to the total number of Tracetree "changes" detected, we compute a cleaned version by assisting Tracetree in three ways. We filter out all changes induced by load balancing, ignore all changes due to violation of the tree hypothesis, and whenever a probe detects a change, we consider that it detects changes in all virtual paths that traverse the changed link (even though they were not directly probed). The result is "Assisted Tracetree."

E. Evaluation With Trace-Driven Simulations

In this section, we use trace-driven simulations to evaluate path rate allocation and sampling techniques.

1) Setup: We build a simulator that takes a dataset with raw traceroutes as input, and for each change in each path, extracts a timestamp and the associated route description. It then simulates how each change tracking technique would probe these paths, complete with their missed changes and estimated (hence inaccurate) feature vectors.

We use our traces described in Section II-B as input for our evaluation. Different monitors in this dataset probe paths at different frequencies. Let $r_{\min}$ be the minimum interval between two consecutive path measurements from a monitor. We set $\lambda_{\max} = 1/r_{\min}$ per-sequence samples per second (the average value over all monitors is 1/190), and this is scaled appropriately for other sampling strategies. This setting is natural in our trace-driven approach: Probing faster than $1/r_{\min}$ is meaningless because the dataset contains no path data more frequent than every $r_{\min}$, and lower $\lambda_{\max}$ would guarantee that some changes would be missed. We set $\lambda_{\min} = 0$ for all monitors.

2) Path Rate Allocation: This section evaluates RL, RL-AGE, MINMISS, and RL-ORACLE using per-sequence, the simplest sampling scheme. Fig. 8(a) shows the fraction of changes missed as a function of $B_{\mathrm{d}}/|\mathcal{D}|$, the detection budget per path. Normalizing per-path facilitates comparison to other datasets. For example, CAIDA's Ark project [4] and DIMES [15] use approximately 0.17×10^{-3} and 8.88×10^{-3} probes per second per virtual path, respectively.

When the budget is too small, not even the oracle can track all changes; whereas in the high budget limit, all techniques converge to zero misses. We see that Ark's probing budget is in the range where even the oracle misses 72% of changes (not shown). To track changes more efficiently, Ark would need more monitors, each tracking a smaller number of paths.

Comparing RL-AGE and RL shows that NN4 reduces the number of missed changes over the simple age-based predictor by up to 47% when the sampling budget is small. For sampling budgets higher than 30×10^{-3}, both RL-AGE and RL perform similarly as most missed changes happen in old, high-prevalence paths where predictors behave similarly. MINMISS reduces the number of missed changes by less than 11% compared to RL. We adopt MINMISS in DTRACK. It is unlikely that we can improve its performance, even if we could it would require a significantly more complex model.

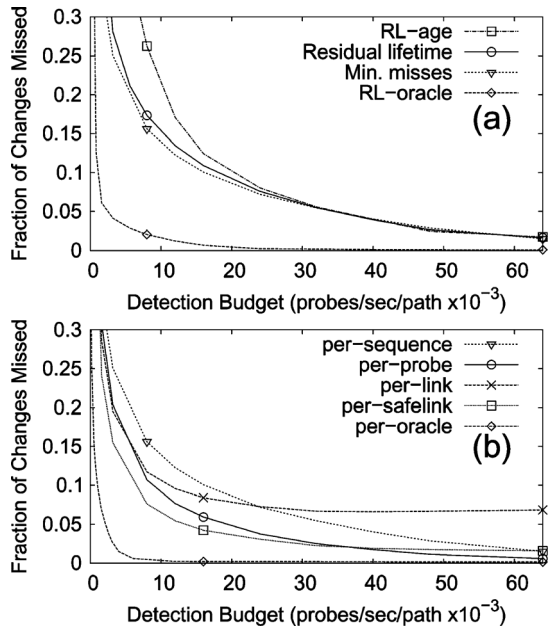


Fig. 8. Fraction of missed changes versus detection budget per path ($B_{\mathrm{d}}/|\mathcal{D}|$): (a) different path sampling rate allocation schemes and per-sequence sampling; (b) different sampling strategies and MINMISS rate allocation.

3) In-Path Sampling: We now use MINMISS as the path rate allocation method and compare the performance of the in-path sampling strategies using Fig. 8(b) ["minimize misses" in Fig. 8(a) and "per-sequence" in Fig. 8(b) are the same].

The per-probe strategy improves on per-sequence by up to 54%. Per-sequence sampling often wastes probes as once a single changed interface is detected, there is no need to sample the rest of the sequence or route; the route can be remapped immediately, and so the search for the next change begins earlier. Per-probe also has a large advantage in spotting short-lived routes, as its sampling rate is $n_{\text{per-sequence}}$ times higher (around 16 times in our data) than per-sequence, greatly decreasing the risk of skipping over them.

Each of per-link and per-probe uses a single probe per sample, but from Fig. 8(b), the latter is clearly superior. This is because the efficiency gains of the sample-sharing strategy of per-link are outweighed by the inherent risks of missed changes (as explained at the end of Section V-C). This tradeoff becomes steadily worse as probing budget increases; in fact for this strategy, the error saturates rather than tending to zero in the limit.

Per-safelink sampling addresses the worst risks of per-link, and over low detection budgets is the best strategy, with up to 28% fewer misses than per-probe. However, at high sampling rates a milder form of the issue affecting per-link still arises, and again the error saturates rather than tending to zero. These results show that exploiting spatial information (like shared links) must be done with great care in the context of tracking, as the very assumptions one is relying on for efficiencies are, by definition, changing (see Tracetree results below).

By default, we use per-safelink sampling in DTRACK, as we expect most deployments to operate at low sampling budgets

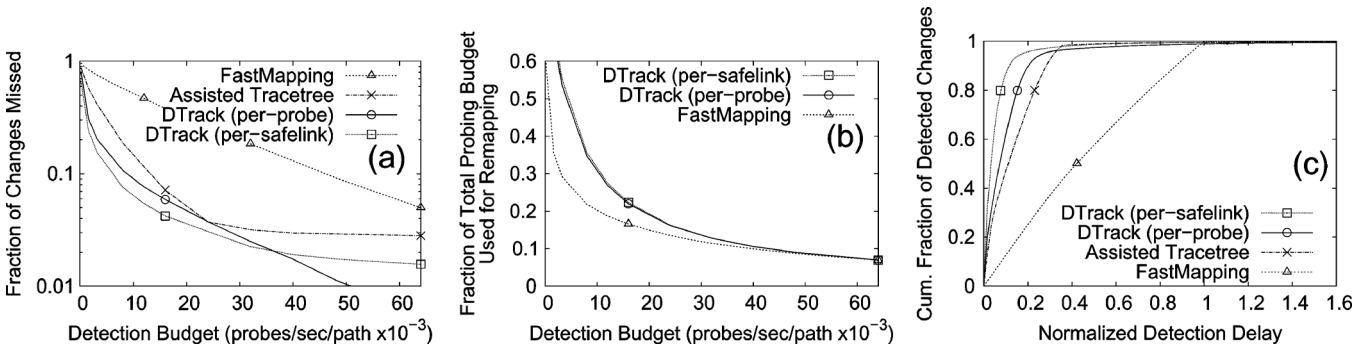


Fig. 9. Comparison of DTRACK, FastMapping, and Assisted Tracetree: (a) fraction of missed changes, (b) fraction of total probing budget used for remapping, and (c) distribution of normalized change detection delay.

(e.g., DIMES and CAIDA's Ark). At very high sampling budgets, we recommend per-probe sampling.

4) Comparing DTRACK *to Alternatives:* Fig. 9(a) replots the per-probe and per-safelink curves from Fig. 8(b) on a logarithmic scale and compares against FastMapping and the assisted form of Tracetree. Each variant of DTRACK outperforms FastMapping by a large margin, up to 89% at intermediate detection budgets. DTRACK also outperforms Assisted Tracetree for all detection budgets, despite the significant degree of assistance provided. We attribute this mainly to the failure of the underlying tree assumption because of load balancing, traffic engineering, and typical AS peering practices. Real (unassisted) Tracetree also suffers from false positives, which in fact grow linearly in probing budget. Already for a probing budget of 8×10^{-3} probes per second per path, Tracetree infers *17 times more* false positives than there are real changes in the dataset.

As an example of the benefits that DTRACK can bring, DIMES, which uses $B_{\mathrm{d}}/|\mathcal{D}| = 8.88 \times 10^{-3}$ probes per second per path, would miss 86% fewer changes (detect 220% more) by using DTRACK instead of periodic traceroutes.

Fig. 9(b) plots the fraction of the total probing budget that is used for remapping, $B_{\mathrm{r}}/B = (B–B_{\mathrm{d}})/B$. We omit Tracetree as it does not perform remapping. At low detection budgets, techniques sample less frequently, and each sample has a higher probability to detect a change (as well as to miss others). In such scenarios, the remapping cost is comparable to the total budget. As the sampling budget increases, the number of changes detected stabilizes, and the remapping cost becomes less significant relative to the total.

Taking again the example of DIMES, even including DTRACK's remapping cost, DIMES would miss 73% less (or detect twice as many) changes using DTRACK instead of periodic traceroutes, while providing complete load balancing information.

Fig. 9(b) allows an operator to compute an initial sampling budget so that DTRACK respects a desired total probing budget B in a real deployment. After DTRACK is running, the operator can readjust the sampling budget as a function of the actual remapping cost found. Also, real deployments can reduce remapping costs compared to the results we show by configuring MDA to use less probes [13].

Fig. 9(c) shows the distribution of the detection delay of detected changes for the different tracking techniques, given a detection budget of $B_{\mathrm{d}}/|\mathcal{D}| = 16 \times 10^{-3}$ probes per second per path. Results for other detection budgets are qualitatively similar. We normalize the detection delay by FastMapping's virtual path sampling period (which is common to all paths).

We see that FastMapping's detection delay is in a sense the worst possible, being almost uniform over the path sampling period. Tracetree samples paths more frequently and achieves lower detection delay. However, both FastMapping and Tracetree, are limited by sampling all paths at the same rate. DTRACK (per-safelink) reduces average detection delay by 57% over FastMapping and has lower delay 99.8% of the time, the exceptions being, not surprisingly, on paths with low sampling budgets.

Low detection delay is important to increase the fidelity of fault detection and tomographic techniques. To see the benefits, say that a monitor uses a total budget B of 64 kb/s to track 8000 paths. It would detect 52% more changes by replacing periodic traceroutes with DTRACK (using safelink), and it would detect 90% of path changes with a delay below 125 s. Replacing classic traceroute by MDA also has the benefit of getting complete and accurate routes.

Our results indicate that DTRACK not only detects more changes, but also has lower detection delay, which should directly benefit applications that need up-to-date information on path stability and network topology.

F. Evaluation in a Real Deployment

We now compare the number of changes detected by FastMapping and DTRACK in a real deployment in PlanetLab. We do not evaluate Tracetree in PlanetLab due to the large number of false positives (Section V-D3).

1) Experimental Setup: We deploy DTRACK and FastMapping on 72 PlanetLab nodes for one week starting March 4, 2011. As in our previous deployments, each monitor selects 1000 destinations at random from a list of 34 820 randomly chosen reachable destinations. DTRACK and FastMapping probe the same destinations. The measurements traverse 7315 ASs and 97% of the large ASs [17].

We set $\lambda_{\max} = 0.05$, which means that DTRACK will sample a path at most once every 20 s: roughly the probing rate required to sample each hop in a path every 5 min. We fix the detection budget B_{d} of both DTRACK and FastMapping at 8 probes per second, which is similar to DIMES' 8.88×10^{-3} probes per second per path and in the range where DTRACK yields the most gains over FastMapping [Fig. 9(a)]. We let the remapping budget B_{r} vary as a function of the number of detected changes.

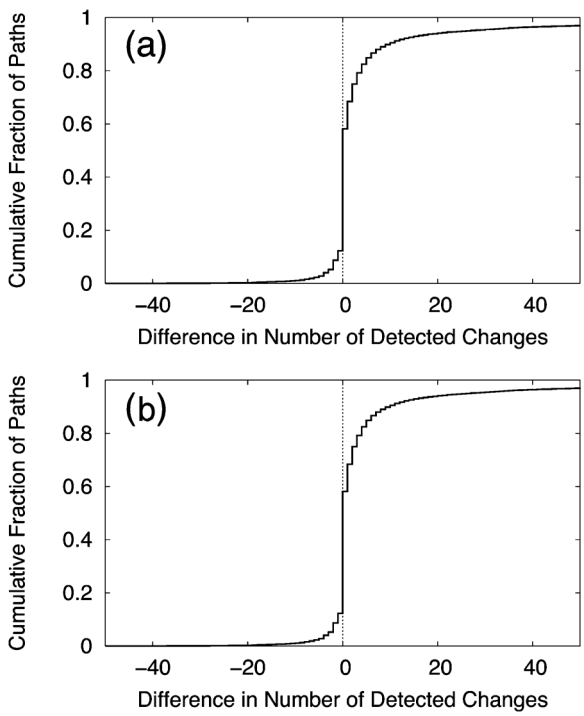


Fig. 10. Evaluation of DTRACK and FastMapping in a real deployment: (a) distribution of the duration of routes where DTRACK and FastMapping missed path changes; (b) difference in number of detected changes (DTRACK minus FastMapping).

We have no ground truth about changes in this deployment. Changes may be detected by DTRACK, FastMapping, or both. If only one technique detected a change, we know the other missed it. Unfortunately, we cannot quantify how many changes DTRACK and FastMapping missed. We use the union of the changes detected by DTRACK and FastMapping as ground truth, similar to how we used our dataset as ground truth in the trace-driven simulations.

2) Tracking Accuracy: During one week of deployment, DTRACK and FastMapping detected a total of 1 280 140 changes. DTRACK detected three times more changes than FastMapping, an increase from the 2.2 times more changes in the trace-driven simulations. The traces used in the trace-driven simulations were collected with FastMapping and limit the number of changes DTRACK can detect.

The remapping cost grows linearly with the number of detected path changes, with an average of 162 probes per path remap. As a result, DTRACK's remapping cost is also three times larger than FastMapping's.

Overall, DTRACK detects 94% and misses 6% of the changes detected by both methods while FastMapping detects 35% and misses 65%. Fig. 10(a) plots the distribution of the duration of the previous route, i.e., the age of the previous route just before it changed, for the sets of path changes missed by one method and detected by the other. We see that most path changes missed by FastMapping happen in short-lived routes, i.e., unstable paths. FastMapping probes all paths at the same rate and misses path changes during instability periods. In contrast, most changes missed by DTRACK happen in routes with long duration, i.e., stable paths. Since DTRACK shifts probing budget from stable to unstable paths, it may take longer to detect and miss changes in stable paths.

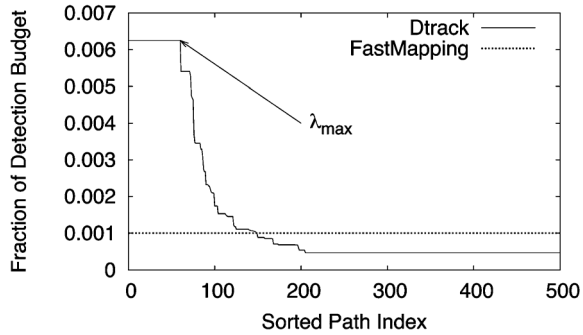


Fig. 11. Snapshot of DTRACK path sampling rates for `chronos.disy.inf.uni-konstanz.de` on March 7, 2011, 6:30 AM.

Fig. 10(b) plots the distribution of the difference of detected changes between DTRACK and FastMapping across the 72 000 monitored paths. It shows that there are a few (12%) paths where FastMapping detects more changes than DTRACK. The difference in the number of detected changes in these paths is usually less than five. Conversely, DTRACK detects more changes than FastMapping in 42% of the paths and can detect 40 changes more than FastMapping for 3.5% of the paths. Although DTRACK may miss some changes, the probing rate after a detection is quickly increased, which reduces subsequent misses substantially. FastMapping, however, systematically misses changes during instability periods.

We illustrate how DTRACK detects more changes in unstable paths without missing many changes on stable paths with a snapshot of path sampling rates allocated by DTRACK at one PlanetLab node at a given point in time (Fig. 11). We see that 60 paths (6%) are considered unstable by NN4 and receive a sampling rate equal to $\lambda_{\max}$, more than six times larger than FastMapping's sampling rate. The higher sampling rate at unstable paths comes at a cost of decreasing the sampling rate of stable paths (paths at $x > 200$) by 56%. Because there are few unstable paths and many stable paths, DTRACK needs to take only a few probes from each stable path to end up with high probing rates on unstable paths. This allows effective tracking in unstable paths without missing many changes in stable paths. Overall, our findings in the real deployment are consistent with our trace-driven simulations; shifting probes from stable to unstable paths allows for more accurate path change tracking.

VI. Related Work

Forwarding Versus Routing Dynamics: Internet path dynamics and routing behavior have captured the interest of the research community since the mid-1990s with Paxson's study of end-to-end routing behavior [9] and Labovitz *et al.*'s findings on BGP instabilities [26]. In this paper, we follow Paxson's approach of using traceroute-style probing to infer end-to-end routes and track virtual path changes. Traceroute is appealing for tracking virtual paths from monitors located at the edge of the Internet for two main reasons. First, traceroute directly measures the forwarding path, whereas AS paths inferred from BGP messages may not match the AS-level forwarding path [27]. Second, traceroute runs from any host connected to the Internet with no privileged access to routers, whereas the collection of BGP messages requires direct access to routers. Although RouteViews and RIPE collect BGP data from some routers

for the community, public BGP data lacks visibility to track all path changes from a given vantage point [28], [29]. When BGP messages from a router close to the traceroute monitor are available, they could help tracking virtual path changes. For instance, Feamster *et al.* [10] showed that BGP messages could be used to predict about 20% of the path failures in their study.

Characterization and Prediction of Path Behavior: Some of the virtual path features that we study are inspired by previous characterizations of Internet paths [9], [10], [12] as discussed in Section II-D. None of these characterization studies, however, use these features to predict future path changes. Although to our knowledge there is no prior work on predicting path changes, Zhang *et al.* [30] studied the degree of constancy of path performance properties (loss, delay, and throughput); constancy is closely related to predictability. Later studies have used routing updates [31], past path performance (for instance, end-to-end losses [32] or round-trip delays [33]), or matrix completion [34] to predict future performance. iNano [35] also "predicts" a number of path properties including PoP-level routes, but their meaning for route prediction is different than ours. Their goal is to predict the PoP-level route of an arbitrary end-to-end path, even though the system only directly measures the route of a small subset of paths. iNano only refreshes measurements once per day and as such cannot closely track path changes. Definition and characterization of new path metrics like Routing State Distance [36] could inspire path features useful for predicting path changes. We plan on evaluating additional features to improve prediction accuracy in the future.

Topology Mapping Techniques: Topology mapping systems [4]–[6], [37] often track routes to a large number of destinations. Many of the topology discovery techniques focus on getting more complete or accurate topology maps by resolving different interfaces to a single router [38]–[40], measuring reverse routes [41], selecting traceroute's sources and destinations to better cover the network [38], [42], [43], using the record-route IP option to complement traceroutes [39], detecting routers performing load balancing [13], or identifying all interfaces on subnets traversed by a path [44]. DTRACK is a good complement to all these techniques. We argue that to get more accurate maps, we should focus the probing capacity on the paths that are changing and also explore spatio-temporal alternatives to simple traditional traceroute sampling. One approach to tracking the evolution of IP topologies is to exploit knowledge of shared links to reduce probing overhead and consequently probe the topology faster as Tracetree [6] and DoubleTree [25] do. As we show in Section V, Tracetree leads to a very large number of false detections. Thus, we choose to guarantee the accuracy and completeness of measured routes by using Paris traceroute's MDA [13]. Most comparable to DTRACK is FastMapping [11]. Section V shows that DTRACK, because of its adaptive probing allocation (instead of a constant rate for all paths) and single-probe sampling strategy (compared to an entire branch of the route at a time), detects up to three times more changes than FastMapping.

VII. Conclusion

This paper presented DTRACK, a path tracking system that proceeds in two steps: path change detection and path remapping. We designed NN4, a simple predictor of path changes that uses as input: route prevalence, route age, number of past route changes, and number of times a route appeared in the past. Although we found that the limits to prediction in general are strong and in particular that NN4 is not highly accurate, it is still useful for allocating probes to paths. DTRACK optimizes path sampling rates based on NN4 predictions. Within each path, DTRACK employs a kind of temporal striping of Paris traceroute. When a change is detected, path remapping uses Paris traceroute's MDA to ensure complete and accurate route measurements. DTRACK detects up to three times more path changes when compared to current traceroute-based tracking techniques, with lower detection delays, and while providing complete load balancer information. DTRACK finds considerably more true changes than Tracetree and none of the very large number of false positives. More generally, we point out that any approach that exploits shared links runs the risk of errors being greatly magnified in the tracking application and should be used with great care.

For future work, we will investigate the benefits of incorporating additional information, such as BGP messages, to increase prediction accuracy, as well as the benefits of coordinating the probing effort across monitors to further enhance probing.

Acknowledgment

This work was performed while D. Veitch was visiting Technicolor.

References

[1] E. Katz-Bassett, H. Madhyastha, J. P. John, A. Krishnamurthy, D. Wetherall, and T. Anderson, "Studying black holes in the Internet with Hubble," in *Proc. USENIX NSDI*, 2008, pp. 247–262.

[2] I. Cunha, R. Teixeira, N. Feamster, and C. Diot, "Measurement methods for fast and accurate blackhole identification with binary tomography," in *Proc. IMC*, 2009, pp. 254–266.

[3] Z. Zhang, Y. Zhang, Y. C. Hu, Z. M. Mao, and R. Bush, "iSPY: Detecting IP prefix hijacking on my own," in *Proc. ACM SIGCOMM*, 2008, pp. 327–338.

[4] K. Claffy, Y. Hyun, K. Keys, M. Fomenkov, and D. Krioukov, "Internet mapping: From art to science," in *Proc. IEEE CATCH*, 2009, pp. 205–211.

[5] Y. Shavitt and E. Shir, "DIMES: Let the Internet measure itself," *Comput. Commun. Rev.*, vol. 35, no. 5, pp. 71–74, 2005.

[6] M. Latapy, C. Magnien, and F. Ouédraogo, "A radar for the Internet," in *Proc. IEEE Int. Conf. Data Mining Workshops*, 2008, pp. 901–908.

[7] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl, "Globally distributed content delivery," *IEEE Internet Comput.*, vol. 6, no. 5, pp. 50–58, Sep.–Oct. 2002.

[8] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris, "Resilient overlay networks," *Oper. Syst. Rev.*, vol. 35, no. 5, pp. 131–145, 2001.

[9] V. Paxson, "End-to-end routing behavior in the Internet," *IEEE/ACM Trans. Netw.*, vol. 5, no. 5, pp. 601–615, Oct. 1997.

[10] N. Feamster, D. Andersen, H. Balakrishnan, and F. Kaashoek, "Measuring the effects of Internet path faults on reactive routing," in *Proc. ACM SIGMETRICS*, 2003, pp. 126–137.

[11] I. Cunha, R. Teixeira, and C. Diot, "Measuring and characterizing end-to-end route dynamics in the presence of load balancing," in *Proc. PAM*, 2011, pp. 235–244.

[12] B. Augustin, T. Friedman, and R. Teixeira, "Measuring load-balanced paths in the Internet," in *Proc. IMC*, 2007, pp. 149–160.

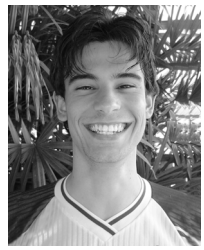
[13] D. Veitch, B. Augustin, T. Friedman, and R. Teixeira, "Failure control in multipath route tracing," in *Proc. IEEE INFOCOM*, 2009, pp. 1395–1403.

[14] J. Friedman and B. Popescu, "Predictive learning via rule ensembles," *Ann. Appl. Stat.*, vol. 2, no. 3, pp. 916–954, 2008.

[15] Y. Shavitt and U. Weinsberg, "Quantifying the importance of vantage points distribution in Internet topology measurements," in *Proc. IEEE INFOCOM*, 2009, pp. 792–800.

[16] Team Cymru, Lake Mary, FL, USA, "IP to ASN mapping," [Online]. Available: http://www.team-cymru.org/Services/ip-to-asn.html

[17] R. Oliveira, D. Pei, W. Willinger, B. Zhang, and L. Zhang, "Quantifying the completeness of the observed Internet AS-level structure," *IEEE/ACM Trans. Netw.*, vol. 18, no. 1, pp. 109–122, Feb. 2010.
[18] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C. N. Chuah, Y. Ganjali, and C. Diot, "Characterization of failures in an operational IP backbone network," *IEEE/ACM Trans. Netw.*, vol. 16, no. 4, pp. 749–762, Aug. 2008.
[19] D. Turner, K. Levchenko, A. Snoeren, and S. Savage, "California fault lines: Understanding the causes and impact of network failures," in *Proc. ACM SIGCOMM*, 2010, pp. 315–326.
[20] Í. Cunha, R. Teixeira, D. Veitch, and C. Diot, "Predicting and tracking Internet path changes," in *Proc. ACM SIGCOMM*, 2011, pp. 122–133.
[21] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, "When is "nearest neighbor" meaningful?," in *Proc. Intl. Conf. Database Theory*, 1999, pp. 217–235.
[22] D. Leonard and D. Loguinov, "Demystifying service discovery: Implementing an Internet-wide scanner," in *Proc. IMC*, 2010, pp. 109–122.
[23] F. Baccelli, S. Machiraju, D. Veitch, and J. Bolot, "The role of PASTA in network measurement," *IEEE/ACM Trans. Netw.*, vol. 17, no. 4, pp. 1340–1353, Aug. 2009.
[24] F. Baccelli, S. Machiraju, D. Veitch, and J. Bolot, "On optimal probing for delay and loss measurement," in *Proc. IMC*, 2007, pp. 291–302.
[25] B. Donnet, P. Raoult, T. Friedman, and M. Crovella, "Efficient algorithms for large-scale topology discovery," in *Proc. ACM SIGMETRICS*, 2005, pp. 327–338.
[26] C. Labovitz, R. Malan, and F. Jahanian, "Internet routing instability," in *Proc. ACM SIGCOMM*, 1997, pp. 205–218.
[27] Z. M. Mao, J. Rexford, J. Wang, and R. H. Katz, "Towards an accurate AS-level traceroute tool," in *Proc. ACM SIGCOMM*, 2003, pp. 365–378.
[28] R. Teixeira and J. Rexford, "A measurement framework for pin-pointing routing changes," in *Proc. SIGCOMM Workshop Netw. Troubleshoot.*, 2004, pp. 313–318.
[29] R. Bush, O. Maennel, M. Roughan, and S. Uhlig, "Internet optometry: Assessing the broken glasses in Internet reachability," in *Proc. IMC*, 2009, pp. 242–253.
[30] Y. Zhang, N. Duffield, V. Paxson, and S. Shenker, "On the consistency of Internet path properties," in *Proc. IMW*, 2001, pp. 197–211.
[31] Y. Zhang, Z. Mao, and J. Wang, "A framework for measuring and predicting the impact of routing changes," in *Proc. IEEE INFOCOM*, 2007, pp. 339–347.
[32] S. Tao, K. Xu, Y. Xu, T. Fei, L. Gao, R. Guerin, J. Kurose, D. Towsley, and Z.-L. Zhang, "Exploring the performance benefits of end-to-end path switching," in *Proc. IEEE ICNP*, 2004, pp. 304–315.
[33] A. Bremler-Barr, E. Cohen, H. Kaplan, and Y. Mansour, "Predicting and bypassing end-to-end Internet service degradations," *IEEE J. Sel. Areas Commun.*, vol. 21, no. 6, pp. 961–978, Aug. 2003.
[34] Y. Liao, W. Du, P. Geurts, and G. Leduc, "Decentralized prediction of end-to-end network performance classes," in *Proc. ACM CoNEXT*, 2011, Art. no. 14.
[35] H. Madhyastha, E. Katz-Bassett, T. Anderson, A. Krishnamurthy, and A. Venkataramani, "iPlane Nano: Path prediction for peer-to-peer applications," in *Proc. USENIX NSDI*, 2009, pp. 137–152.
[36] G. Gürsun, N. Ruchansky, E. Terzi, and M. Crovella, "Routing state distance: A path-based metric for network analysis," in *Proc. IMC*, 2012, pp. 239–252.
[37] H. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani, "iPlane: An information plane for distributed services," in *Proc. USENIX OSDI*, 2006, pp. 367–380.
[38] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP topologies with rocketfuel," in *Proc. ACM SIGCOMM*, 2002, pp. 133–145.
[39] R. Sherwood, A. Bender, and N. Spring, "DisCarte: A disjunctive Internet cartographer," in *Proc. ACM SIGCOMM*, 2008, pp. 303–314.
[40] J. Sherry, E. Katz-Bassett, M. Pimenova, H. Madhyastha, A. Krishnamurthy, and T. Anderson, "Resolving IP aliases with prespecified timestamps," in *Proc. IMC*, 2010, pp. 172–178.
[41] E. Katz-Bassett, H. Madhyastha, V. Adhikari, C. Scott, J. Sherry, P. van Wesep, A. Krishnamurthy, and T. Anderson, "Reverse traceroute," in *Proc. USENIX NSDI*, 2010, p. 15.
[42] R. Beverly, A. Berger, and G. Xie, "Primitives for active Internet topology mapping: Toward high-frequency characterization," in *Proc. IMC*, 2010, pp. 165–171.
[43] X. Fan and J. Heidemann, "Selecting representative IP addresses for Internet topology studies," in *Proc. IMC*, 2010, pp. 411–423.
[44] E. M. Tozal and K. Sarac, "TraceNET: An Internet topology data collector," in *Proc. IMC*, 2010, pp. 356–368.



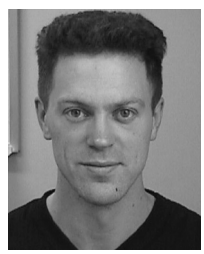
Ítalo Cunha received the B.Sc. and M.Sc. degrees from the Universidade Federal de Minas Gerais, Belo Horizonte, Brazil, in 2004 and 2007, respectively, and the Ph.D. degree from UPMC Sorbonne Universités, Paris, France, in 2011, all in computer science.

He worked on Internet troubleshooting with the University of Washington, Seattle, WA, USA, during his Ph.D. studies and performed postdoctoral research with Technicolor Research and Innovation, Paris, France. He is currently an Assistant Professor with the Computer Science Department, Universidade Federal de Minas Gerais. His research interests are in large-scale distributed systems and network monitoring and troubleshooting.

Renata Teixeira received the B.Sc. degree in computer science and M.Sc. degree in electrical engineering from the Universidade Federal do Rio de Janeiro, Brazil, in 1997 and 1999, respectively, and the Ph.D. degree in computer science from the University of California, San Diego, CA, USA, in 2005.

During her Ph.D. studies, she worked on Internet routing with AT&T Research, Florham Park, NJ, USA. She is currently a Researcher with the Centre National de la Recherche Scientifique (CNRS), LIP6, UPMC Sorbonne Universites, Paris, France. She was a Visiting Scholar with the University of California, Berkeley/ICSI, Berkeley, CA, USA, in 2011. Her research interests are in measurement, analysis, and management of data networks.

Dr. Teixeira serves in the Editorial Board of the IEEE/ACM TRANSACTIONS ON NETWORKING and the *ACM SIGCOMM Computer Communication Review*. She is also a member of the steering committee of the ACM Internet Measurement Conference and has been active in the program committees of ACM SIGCOMM, ACM IMC, PAM, and IEEE INFOCOM, among others.



Darryl Veitch (M'98–SM'02–F'10) received the B.Sc. Hons. degree from Monash University, Melbourne, Australia, in 1985, and the Ph.D. degree in mathematics from Cambridge University, Cambridge, U.K., in 1990.

He worked with TRL, Telstra, Melbourne, Australia; CNET, France Telecom, Paris, France; KTH, Stockholm, Sweden; INRIA, Sophia Antipolis and Paris, France; Bellcore, Red Bank, NJ, USA; RMIT, Melbourne, Australia; Technicolor, Paris, France; and EMUlab and CUBIN at The University of Melbourne, Melbourne, Australia, where he is currently a Professorial Research Fellow. His research interests are in computer networking and include traffic modeling, parameter estimation, the theory and practice of active measurement, traffic sampling and sketching, and clock synchronization over networks.

Christophe Diot received the Ph.D. degree in computer science from INP Grenoble, Grenoble, France, in 1991.

With INRIA Sophia-Antipolis, Sophia-Antipolis, France, from 1993 to 1998, he pioneered diffserv, single-source multicast, and peer-to-peer online games. With Sprint, San Francisco, CA, USA, from 1998 to 2003, he introduced large-scale Internet measurements. He was with Intel Research, Cambridge, U.K., from 2003 to 2005, and then joined Thomson, where he started two research labs: Paris, France, in 2006, and Palo Alto, CA, USA, in 2011. Thomson became Technicolor in 2010. He is currently a Chief Scientist with Technicolor. He has around 20 patents and more than 300 publications in major conferences and journals. His research activities now focus on content delivery architectures, recommender systems, and personalization.

Dr. Diot is a Fellow of the Association for Computing Machinery (ACM).